MANAGING FIRMWARE DOWNLOAD

FIELD OF THE INVENTION

The present invention relates generally to managing download of firmware and functions implemented by firmware.

BACKGROUND OF THE INVENTION

Multi-function devices may comprise embedded firmware. A processor may be used to implement the functionality of the embedded firmware, wherein for each function, there may be an independent host driver. Host drivers may be required to download the firmware images before being able to use the functions.

An operating system, which is used to operate drivers connected to chips that implement the multi-function device, may load the drivers in an arbitrary order for each function. It may not be known in advance which driver of a function will be loaded first. If more than one driver should download the same embedded firmware inside the multi-function device (or chip), then an already downloaded firmware, which is already running, may be overwritten and reinitialized, thereby interrupting operation of that function. To complicate matters, each driver may implement different functionality, and the common firmware, which it is desired to download, may have to be updated accordingly.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood and appreciated more fully from the following detailed description taken in conjunction with the appended drawings wherein:

Fig. 1 is a simplified flow chart of a method for managing firmware download in accordance with an embodiment of the invention; and

Figs. 2A and 2B are a simplified block diagram of apparatus and a system for managing firmware download in accordance with an embodiment of the invention.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

Reference is now made to Figs. 1, 2A and 2B, which illustrate a method, apparatus and system for managing downloading of firmware functions in accordance with an embodiment of the invention.

A multi-function device 10 may comprise at least two firmware functions 12 which are accessible by more than one driver or by more than one access operation of the same driver 14. A "firmware function" refers to a function that may use firmware as at least part of its implementation; the function may also use dedicated hardware. A processor 16 may be adapted to manage downloading two or more firmware functions 12. In the system, there may be a memory 18 in communication with processor 16. The processor 16 may be adapted to reduce a risk of at least one of the drivers 14 overwriting firmware that has been downloaded and is being used by another of the drivers 14. The processor 16 may be adapted to download at least two firmware functions 12 with a single download.

Memory 18 may comprise without limitation an external electrically erasable, programmable read only memory (EEPROM), or an on-board factory programmable set of jumpers, for example, containing data about firmware functionality.

At least one of the drivers 14 may be initialized (step 101) with information to determine a desired firmware sufficient to implement a desired functionality. The driver 14, also referred to as a host driver 14, may read the memory 18 (step 102) during the initialization sequence to determine the desired firmware to implement the functionality for itself and for the other functions, such as but not limited to, the required chip functionality and which firmware modules need to be downloaded.

The driver 14 may verify if the desired firmware has been downloaded by another driver (step 103). This may be accomplished without limitation, for example, by

the driver 14 checking a register (or a bit) accessible by all functions (e.g., *FW_DL_done*), to check if the firmware download was already done by one of the other functions. This register may appears in the function's register space and may be accessible for read and writes by the functions.

If the firmware has been downloaded already, then execution of the function's firmware may commence (step 111) and the download process may end (step 112). If the firmware has not been downloaded already by another function, then the desired firmware may be downloaded (step 103A).

The driver 14 may lock the access to the device 10 (step 104). Locking the device 10 may be implemented in several ways, such as but not limited to:

a. a standard memory spin lock, which may require that the drivers spin lock on a memory location common to all of the drivers;

b. PCI (peripheral component interface) bus locking on a memory location of the driver, which may require a PCI bus lock to be implemented in the host driver 14 and device 10; or

c. locking a device memory register, such as but not limited to, a "set by read" register as a semaphore or a "set by write" register as a semaphore.

If the driver 14 does not lock access to device 10, then the procedure may go back to step 103. If the driver 14 locks access to device 10, and thereby gains access to the device 10 for download purposes, the driver 14 may again verify that it is the only driver that is accessing the device 10 for download purposes (step 105), and that the firmware of device 10 has not yet been downloaded, so as to avoid race conditions (step 106). If the firmware of device 10 has already been downloaded, then the process may skip to step 109, described hereinbelow. If the firmware of device 10 has not yet been

downloaded, then the driver 14 may commence downloading the firmware (step 107). Downloading may comprise without limitation using programmed I/O, memory-to-memory copy, or bus master DMA (direct memory access) transactions, for example.

The method may comprise downloading at least two firmware functions with a single download. Firmware may be downloaded that is common to at least two of the drivers.

At the end of the download, the driver 14 may set a register or flag (e.g., *FW_DL_done*) informing that the desired firmware has been downloaded (step 108). The driver 14 may set another bit (e.g., per function, in the function's register space) to start the firmware thread implementing the firmware function (step 109). A functionally may be implemented that is common to at least two of the drivers. Alternatively, a functionality for one of the drivers may be implemented that is different than another of the drivers.

If another function's driver sees that the lock is not gained, that driver may spin lock, wait to check for the lock again or wait until the firmware load is done by the function that is currently locking it (step 109A) and then start its own function's firmware execution (step 111). At the end of the download process, the host driver 14 may release the lock (step 110) according to the lock mechanism used, and permit access to the desired firmware by drivers other than the host driver 14.

It will be appreciated by persons skilled in the art that the present invention is not limited by what has been particularly shown and described herein above. Rather the scope of the invention is defined by the claims that follow: